CogniSight SDK
User's Manual

API INTERFACING TO A NEUROMEM SILICON NETWORK

Version 2.8
Revised 10/10/2015

CogniSight SDK is a product of General Vision, Inc. (GV)

This manual is copyrighted and published by GV. All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of GV.

For information about ownership, copyrights, warranties and liabilities, refer to the document Standard Terms And Conditions Of Sale or contact us at www.general-vision.com.

TABLE OF CONTENTS

Table of Contents 2
Introduction 4
Compatible hardware 4
Getting Started 4
Files and Folders 4
Review the Examples 4
Data type definitions 5
Image Frame 5
Region(s) of Interest 5
Region(s) of Search 6
Visual Objects 6
Knowledge 6
Project 6
Functions overview 6
Dichotomy of Possible applications 7
Connecting to hardware 8
int Version(); 8
Void SetPlatform(int platform); 8
int Connect(int DeviceID); 8
int Disconnect(); 8
int CountNeuronsandReset(); 8
Image management 9
void BufferToCS(unsigned char *imageBuffer, int Width, int Height, int BytesPerPixel) 9
void GetCSBuffInfo(int *Width, int *Height, int *BytePerPixel) 9
void CSToBuffer(unsigned char *imageBuffer) 9
Feature Extractions 10
void SetROI(int Width, int Height) 10
void GetROI(int *Width, int *Height) 10
void SetFeatParams(int FeatID, int Normalize, int Minif, int Maxif, int Param1, int Param2) 10
void GetFeatParams(int *FeatID, int *Normalize, int *Minif, int *Maxif, int *Param1, int *Param2) 11
int GetFeature(int X, int Y, unsigned char *Vector) 11
Search Parameters 11
void SetROS(int Left, int Top, int Width, int Height) 11
void GetROS(int *Left, int *Top, int *Width, int *Height) 11
void SetScanParams(int *stepX, int *stepY, int *skipX, int *skipY) 11
void GetScanParams(int *stepX, int *stepY, int *skipX, int *skipY) 11
int GetROSVectors(int stepX, int stepY, unsigned char *Vectors, int *VLength) 11
Learning functions 12
int LearnROI(int X, int Y, int Category) 12
int LearnROS(int stepX, int stepY, int category) 12

int BuildROSCodebook(int stepX, int stepY, int CatAllocMode) 12
void FindSalientBlocks(int K, int *CtrX, int *CtrY, int *AIF) 12
int ImageToNeurons(int stepX, int stepY, int UsePositionAsContext) 14
Recognition functions 14
int RecoROI(int X, int Y, int K, int *distance, int *category, int *nid) 14
int RecognizeROI(int X, int Y, int K, int *distances, int *categories, int *nids) 14
int FindROSObjects(int stepX, int stepY, int skipX, int skipY, int *Xpos, int *Ypos, int* distance, int* category, int* nid) 15
int FindROSAnomalies(int stepX, int stepY, int MaxNbr, int *Xpos, int *Ypos) 15
int MapROS(int stepX, int stepY, int *CatMap, int *DistMap, int *NidMap) 15
Knowledge Level Functions 15
int SaveProject(char *filename) 15
int LoadProject(char *filename) 15
int ClearNeurons(); 16
Getting closer to the neurons 17
Appendix A: Feature Extractions 18
SubSample 18
Histogram 18
Histogram Cumulative 18
SubsampleRGB 18
Histogram RGB 19
Histogram RGB Cumulative 19
Appendix B: Tutorial, Hints and Tips 20
Supervised Object Learning 20
Using multiple feature extractions for object classification 21
What if an object appears at different scale factors? 21
Dealing with objects of different sizes 23
Surface Inspection 24

INTRODUCTION

The CogniSight SDK is a software development kit for image learning and recognition based on a NeuroMem neural network which is an essential component for the high-speed pattern classification abundantly needed in machine vision and video analytics. The functions of the CogniSight SDK lets you teach objects or textures from reference images and monitor the throughput and accuracy of the recognition against new images. CogniSight can map information from images through feature extractions based on contrast, shape, color, texture and more. It does not require any procedural computing (no programming), but an interface to select and teach examples and consolidate results from the multiple neural networks for a final classification. The search functions can report the list of recognized objects with their category and a confidence factor, or the list of anomalies or novelties. Transform functions can produce distribution map per category or distance and best-match compression. To increase the accuracy of a recognition engine, you can train multiple sub-networks associated to different sensor inputs and/or different feature extractions. The final recognition can then build a more robust decision by consolidating the response of these multiple networks and weighting cases of unknown and uncertainty.

Compatible hardware

The CogniSight SDK interfaces seamlessly with a single NeuroStack board or a stack of multiple NeuroStack boards demonstrating the speed and high scalability of the CogniSight technology.

If a compatible hardware is detected, the API uses the neurons of the CM1K chip(s) to learn and recognize vectors. Beyond these operations the configuration of the hardware will determine if it can execute other tasks programmed in its FPGA or if these tasks must be executed on the PC host.

In the absence of compatible hardware, the API can emulate a cycle accurate simulation of a single CM1K chip or 1024 neurons.

GETTING STARTED

Files and Folders

- Bin folder
    - Header files and DLLs compiled for 32-bit and 64-bit operating systems
        - GVEngine.dll
        - GVAPI.h, header file with the declaration of the functions
        - GVREG.h, file with the declaration of the CM1K registers
- Examples, Images and Projects folders

Review the Examples

- The examples of this SDK illustrate different use models of a CogniSight engine, including learning and recognition methods for discrete visual objects as well as surfaces.
- The examples are a sound base to write functions calling the DLL
- Images and default project files are supplied in the Images folder for your convenience.

Data type definitions

A CogniSight engine manipulates the following components:

- Image Frame
- The "What" or Region of Interest (ROI)
- The "Where" or Region of Search (ROS)
- The Knowledge or content of the neurons (KN)

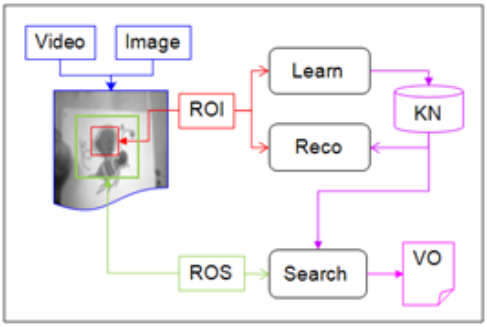
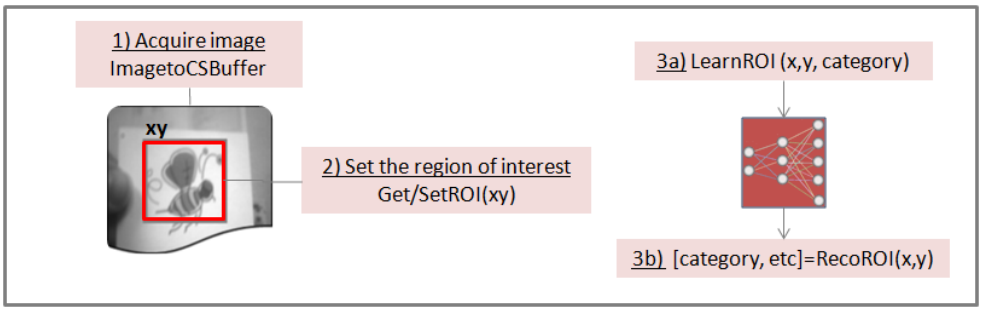


IMAGE FRAME

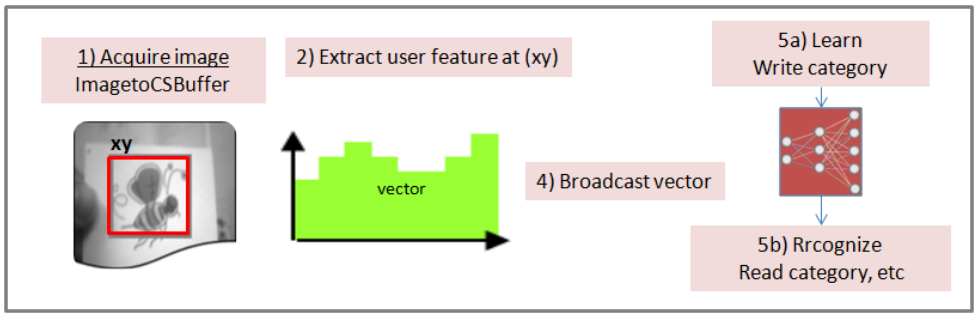
When an image is loaded from an image file or a movie file, it must be transferred to the CogniSight image memory frame so the engine can access the pixel values for feature extraction, learning and recognition. This transfer is not necessary if the image has been acquired by a CogniSight Sensor in which case it resides automatically in the CogniSight image memory frame.

REGION(S) OF INTEREST

A Region Of Interest (ROI) is the primitive area to learn or recognize. It can be a discrete object, part of an object, a significant feature in a scene, a patch of texture, etc. From the pixel values inside an ROI, the CogniSight engine extracts a signature. This signature becomes the feature **vector** learned or recognized by the neurons.



The CogniSight API includes a selection of pre-defined feature extractions, and the diagram below illustrates the internal steps of the LearnROI and RecognizeROI functions.



The classification of the ROI can be based on a single or multiple features (or signatures). In the case of multiple features, the context of the neurons must be changed for each type of feature. The context value can be assigned to a feature identification number for example starting at the value 1 (value 0 is reserved to activate neurons of all contexts at once). For example, the feature subsample can be assigned to context 1 and the feature histogram can

be assigned to context 2. The proper context has to be activated prior to broadcasting the corresponding feature vector.

Also, if an application must classify more than one family of objects at a time (i.e. (1) filling level of a bottle and (2) quality of its front label), it shall store the definition of the different ROIs and make sure to change the Global Context of the neurons when switching between ROIs. For example the Context #1 will be assigned to the neurons taught with examples of filling levels and the Context #2 to the neurons taught with examples of labels. For more information about the usage of multiple ROIs refer to Appendix C.

REGION(S) OF SEARCH

The Region of Search **(ROS)** is the area to observe through the "aperture" of a given ROI and using the knowledge of the neurons.

VISUAL OBJECTS

The Visual Objects **(VO)** are a list of identified locations with their recognized category as a result of a Search over a region of scan. They can be presented as an array or in Transform images showing their spatial distribution based on attributes such as their category or their similarity factor.

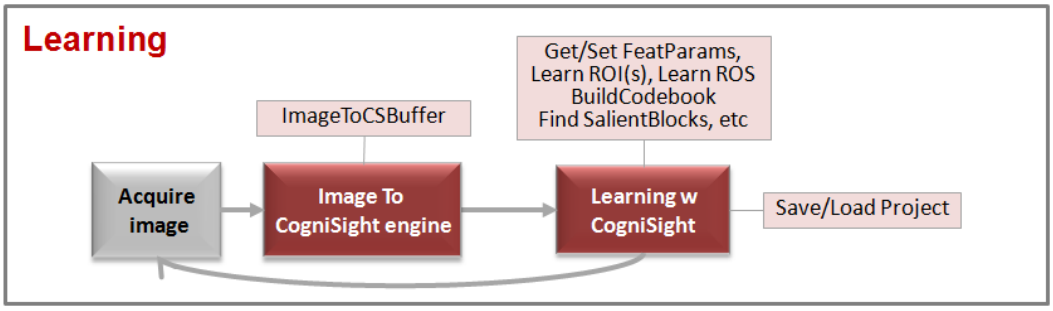
KNOWLEDGE

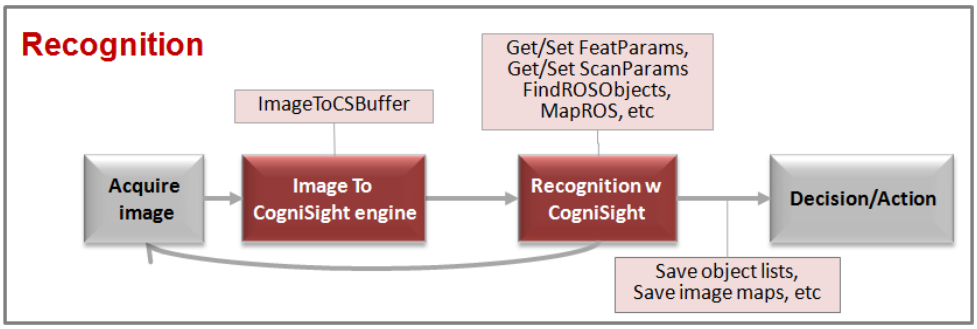
The knowledge file **(KN)** is generated by the NeuroMem network as it learns examples of visual objects or patterns extracted from the images and submitted by the CogniSight engine to the neurons. It can be tuned and enriched over time with new examples. Depending on the application, the selection of the objects to learn can be supervised or unsupervised. The knowledge can be built using single or multiple feature vectors characterizing an object or texture.

PROJECT

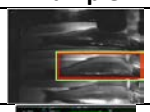
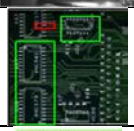
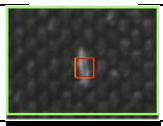
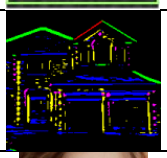
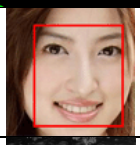
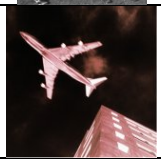
A project file is more complete than just a knowledge file because it describes exactly how the knowledge was built (the network settings, the experts and their feature extractions, the user preferences such as the names of the categories and their color tags). The project file includes all the necessary information to expand an existing knowledge and to use it for immediate recognition of still or live images, produce meta data, selective recording and more.

Functions overview





Dichotomy of Possible applications

Application	ROI	ROS	Decision rules	Example
Simple part inspection	Single	Single	Best match category	
Complex part inspection	Multiple	Multiple	Decision rule on Best Matches	
OCR/Kanji Character	Single	Single	Sequence of the Best Matches	
Surface anomaly	Single	Single, Multiple	Unknown locations	
Edge detection	Single	Single	Map of the Best Matches	
Cooperative Face Recognition	Mutiple	Single	Decision rule on Best Matches	
Face detection in a scene	Multiple	Single	Sequence of the Best Matches	
Target Tracking	Single	Single, adaptive	Center of the Best Matches	

CONNECTING TO HARDWARE

int Version();

Returns the version of the DLL. The last digit represents the platform 0= Simulation or 1= NeuroStack.

Void SetPlatform(int platform);

Set the hardware platform: 0= Simulation, 1=NeuroStack

int Connect(int DeviceID);

Establishes communication with the CogniSight device. This function returns 0 if the connection is successful.
In the case of a USB device, DeviceID is the USB device number connected to your host. Default is 0.

int Disconnect();

Closes the communication with the current device.

int CountNeuronsandReset();

Detect the number of neurons available in the connected platform and then clear them all including their registers and memory.

Remark 1: Detecting the neurons requires to commit them all. As a consequence, its execution will overwrite the existing content of the neurons if any if committed.

Remark 2: The execution of this function can take several seconds depending on the size of the chain of CM1K chips plus it erases the contents of the neurons.

Pursuant to the two previous remarks, it is highly recommended to use CountNeuronsandRset only at the launch of an application.

IMAGE MANAGEMENT

When an image is loaded from an image file or a movie file, it must be transferred to the CogniSight image memory frame so the engine can access the pixel values for feature extraction, learning and recognition.

The image width is equal to the number of pixels per column and its height equal to the number of rows of pixels. Depending on the format of the source image, the number of bytes encoded per pixel can vary. The CogniSight image memory frame supports 1 byte per pixel in the case of a monochrome image or 3 bytes per pixel in the case of a color image where the 3 bytes representthe pixel's R, G, B intensities

void BufferToCS(unsigned char *imageBuffer, int Width, int Height, int BytesPerPixel)

Load a byte array into a memory frame of the CogniSight workspace with a size equal to Width * Height * Bytes Per Pixel.

void GetCSBuffInfo(int *Width, int *Height, int *BytePerPixel)

Reads the parameters Width, Height and BytePerPixel describing the current CogniSight memory frame.

void CSToBuffer(unsigned char *imageBuffer)

Read the CogniSight memory frame and returns it as a byte array. The format of this byte array is described by the CSBuffInfo parameters Width, Height and BytePerPixel. This function can be useful to retrieve an image which was directly acquired on a compatible hardware with on-board sensor.

FEATURE EXTRACTIONS

The Region Of Interest (ROI) is the primitive area to learn or recognize. It can be a discrete object, part of an object, a significant feature in a scene, a patch of texture, etc. From the pixel values inside an ROI, the CogniSight engine extracts a signature which becomes the feature **vector** learned or recognized by the neurons.

void SetROI(int Width, int Height)

Defines the nominal width and height of the current ROI.

void GetROI(int *Width, int *Height)

Reads the nominal width and height of the current ROI.

void SetFeatParams(int FeatID, int Normalize, int Minif, int Maxif, int Param1, int Param2)

Defines the current feature extraction and learning settings.

FeatID	Identifier of the feature extraction (see table below)
Normalize	Flag indicating if the amplitude of the feature vector shall be normalized between [0-255]
Minif	Minimum influence field of a newly committed neuron
Maxif	Maximum influence field of a newly committed neuron
Param1	1st parameter of the feature extraction (if applicable)
Param2	2nd parameter of the feature extraction (if applicable)

FeatID	Description	Param1	Param2
0	Monochrome subsample or average intensity of up to 256 blocks fitting inside the ROI	Block width	Block height
1	Grey histogram Or the number of pixels per 256 grey-level values.	n/a	n/a
2	Grey histogram cumulative	n/a	n/a
3	Color subsample or average Red, Green and Blue intensities of up to 85 blocks fitting inside the ROI	Block width	Block height
4	Color histogram Or sequence of the red, green and blue histograms, each of 85 values.	n/a	n/a
5	Color histogram cumulative	n/a	n/a

Refer to the Appendix A for more details on the feature extractions.

void GetFeatParams(int *FeatID, int *Normalize, int *Minif, int *Maxif, int *Param1, int *Param2)

Reads the current feature extraction and learning settings (see above description)

int GetFeature(int X, int Y, unsigned char *Vector)

Returns the feature vector extracted from the ROI at the (X,Y) location in the image.

SEARCH PARAMETERS

void SetROS(int Left, int Top, int Width, int Height)

Defines the current Region Of Search which can range from the size of the current ROI to the entire image.

void GetROS(int *Left, int *Top, int *Width, int *Height)

Reads the current Region Of Search

void SetScanParams(int *stepX, int *stepY, int *skipX, int *skipY)

Defines how the region of search will be scanned.

stepX	Displacement of the region for the next learning or recognition along the horizontal axis
stepY	Displacement of the region for the next learning or recognition along the vertical axis
skipX	Option to move by an entire region's width if the region is recognized
skipY	Option to move by an entire region's height if the region is recognized

The scanning parameters help compromise between the level of spatial details in the results and the execution time of the scanning function. The smaller the step the more region to process but the finer the details.

void GetScanParams(int *stepX, int *stepY, int *skipX, int *skipY)

Reads the current scanning settings (see above description)

int GetROSVectors(int stepX, int stepY, unsigned char *Vectors, int *VLength)

Returns the list of feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY. VLength reports the length of the feature vectors which is function of the featID and its parameters in use. Vectors is an array with a dimension of VLength times Number_of_Steps covered during the scanning.

LEARNING FUNCTIONS

int LearnROI(int X, int Y, int Category)

Learns the feature vector extracted from the ROI at the (X,Y) location in the image as Category. Category can range between [0, 32,738]. The category 0 is reserved to learn a “background” or “discard” example.

int LearnROS(int stepX, int stepY, int category)

Learns the feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY. All these vectors are assigned the same user-defined category.

int BuildROSCodebook(int stepX, int stepY, int CatAllocMode)

Learns the feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY. All the vectors representing novelty are automatically learned and assigned a category value incremented each time a novelty is detected by the neurons.
CatAllocMode: Defines which category to assign to a block which is not recognized by the currently committed neurons:

- 0: constant value
- 1: auto-increment by 1
- 2: maximum delta between the vectLen components
- 3: average value of the vectLen components
- 4: index of the vector committing the neuron. This information can be used to retrieve the XY origin of the vector in an image, or else.

Note that the function does not clear the knowledge and uses current Minif, Maxif, GCR. It returns the number of committed neurons.

void FindSalientBlocks(int K, int *CtrX, int *CtrY, int *AIF)

Learn the adjacent blocks composing the entire image and returns the center position of the blocks which have committed the neurons with the K largest Influence fields.

The higher the influence field of a neuron, the more salient the pattern held in its memory since we can assume that this neuron is among the least which had to shrink their influence field during the learning due to similar blocks.

Remark1: The saliency is relative to the overall content of the image.

Remark2: Adjacent blocks belonging to a uniform background area have similar feature vectors. Learning them with a different category forces the commitment of one neuron per block but their influence fields are very small, if not set to the minimum, since the patterns are redundant.

Remark3: At the end of the function, the contents of the neuron is still the segmentation of the image into blocks.

Example of the top10 salient blocks in lena.jpg:

The size of the block is the size of the current ROI and its default should be a block of 16x16 pixels, but it is possible to execute the function on larger or smaller block size.

The feature used during the learning is the current feature extraction. Its default should be a monochrome or color subsample (featID=0 or featID=3), but the function will actually execute with any feature extrcation.

Warning, this function clears the content of the neurons prior to its execution.

int ImageToNeurons(int stepX, int stepY, int UsePositionAsContext)

Load the feature vector of each ROI into the neurons using a raster displacement and incremental categories so each ROI commits a neuron. If the feature extraction is a subsample of 16x16, this means that the neurons hold the entire image divided into tiles of 16x16 pixels.

The resulting neurons have their category equal to their identifier. They are both equivalent to an encoding of the XY coordinates of the model in the image.

The function clears the existing knowledge, but restores the global registers upon termination such as the NSR, MAXIF and GCR. It returns the number of neurons holding a row of pixel data. This number depends on the image size, ROI size and Step. It is useful to convert a neuron identifier into a pixel location in the original image.

Option to use the position of the input block to encode a context. This can be useful to limit the firing patterns to vertical rows and columns in recognition.
0= uses the current context
1= set context to horizontal position of the block (modulo 127 since the context must range between [1,127])
2= set context to vertical position of the block (modulo 127 since the context must range between [1,127])

The function can be executed in LR or SRmode If SR mode, set the default Maxif to a significant value prior to execution.

RECOGNITION FUNCTIONS

int RecoROI(int X, int Y, int K, int *distance, int *category, int *nid)

Recognizes the feature vector extracted from the ROI at the (X,Y) location in the image and reports the distance, category and neuron identifier of the closest neuron. The function returns the value of the Network Status Register (NSR).

If bit 3 of the NSR, it means that there is no uncertainty regarding the recognition of the ROI.

If bit 2 of the NSR is equal to 1, it means that there are additional neurons which recognize the ROI with a different category and possibly farthest distance. Their response can be read with successive register access (Read DIST, Read CAT, Read NID).

int RecognizeROI(int X, int Y, int K, int *distances, int *categories, int *nids)

Recognizes the feature vector extracted from the ROI at the (X,Y) location in the image and reports the distance, category and neuron identifier of the K closest neurons, if any. The function returns the value R equal to the number of positive responses or K whichever is smaller.

K can range from 1 to the number of committed neurons.
The three output arrays distances, categories and nids have a dimension equal to K, but the significant information is stored in the R first values, with R being the number of positive responses. R is lesser than or equal to K.
Beyond the Rth index, the values reported in the output arrays are 65535 or 0xFFFF.

For example, if K=2 and only one neuron fires, distances[0]= distance of the feature vector to the single firing neuron. Distances[1]=65535. Same for the categories and nids arrays.

int FindROSObjects(int stepX, int stepY, int skipX, int skipY, int *Xpos, int *Ypos, int* distance, int* category, int* nid)

Recognizes the feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY. The output arrays describe all the recognized objects including their X and Y position in the image, and the distance, category and neuron identifier of the closest firing neuron. The skip_if_reco flag allows to optimize the scanning process by stepping directly to the next adjacent ROI in the case of a positive recognition.

int FindROSAnomalies(int stepX, int stepY, int MaxNbr, int *Xpos, int *Ypos)

Reports the feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY and which are not recognized. The output arrays Xpos and Ypos report the positions of these unknown locations.

int MapROS(int stepX, int stepY, int *CatMap, int *DistMap, int *NidMap)

Builds the lists of the categories, distances and neurons' identifiers recognized at each step of a raster displacement within the Region of Search.
The output arrays CatMap, DistMap and NidMap can then be mapped into images with a width of MapWidth for a 2D mapping of the categories, distances and firing neurons in the region of search. The amplitude of these values can be much greater than intensity values ranging between [0-255] and scaling might be required.

The values of the CatMap can range between [0,32,364].
The values of the DistMap can range between [0, 65535].
The values of the NidMap can range between [0, Number of neurons available].

The function returns the width of the resulting map, or the number of samples which constitute a row. This number is useful to display the output array in a 2D plane.

KNOWLEDGE LEVEL FUNCTIONS

int SaveProject(char *filename)

Saves to file the knowledge stored in the neurons as well as the current ROI size, the feature extraction and its parameters and the settings of the neural network (maxif, minif);

int LoadProject(char *filename)

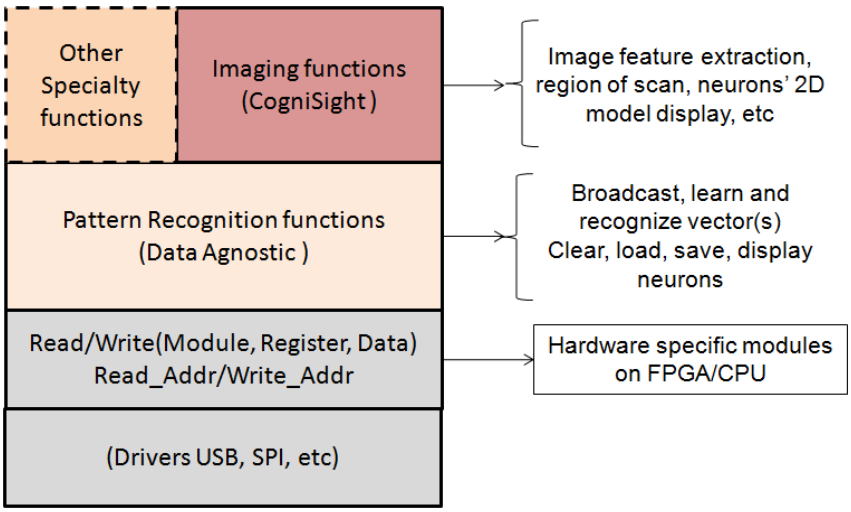
Restores from file the content of the neurons as well as the current ROI size, the feature extraction and its parameters and the settings of the neural network (maxif, minif);

int ClearNeurons();

Clear the entire contents of all the neurons including registers and memory cells. The function returns the number of committed neurons which should be equal to zero.

GETTING CLOSER TO THE NEURONS

The CogniSight SDK is developed on top of the NeuroMem SDK as described in the diagram below.


Other Specialty functions
Imaging functions (CogniSight)
Image feature extraction, region of scan, neurons' 2D model display, etc
Pattern Recognition functions (Data Agnostic)
Broadcast, learn and recognize vector(s) Clear, load, save, display neurons
Read/Write(Module, Register, Data) Read_Addr/Write_Addr
Hardware specific modules on FPGA/CPU
(Drivers USB, SPI, etc)

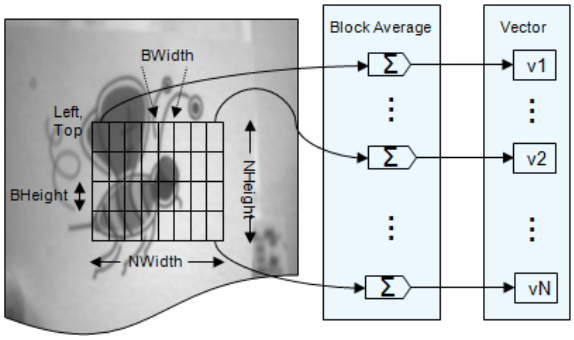
The functions of the SDK which are agnostics to the data types and do not necessarily apply to vectors extracted from images are described in the documentation of the NeuroMem SDK.

The communication protocol and register level access are described in the documentation of the NeuroMem SDK.

APPENDIX A: FEATURE EXTRACTIONS

SubSample

Subsample is a vector which appends the average intensity of blocks of pixels extracted from the region of interest. The region must contain less than 256 blocks so the output vector fits on 256 bytes. The blocks are all the same size, but not necessarily square. They are surveyed in a raster displacement and their average intensity is assembled into vector.



- The region with a size [NWidth, NHeight] is divided into up to 256 blocks of size [BWIDTH, BHEIGHT].
- The pixels of block #i are averaged to produce the i^{th} component of the signature vector.
- The relationship between the four parameters is :
    - NWIDTH= n*BWIDTH
    - NHEIGHT= m*BHEIGHT
    - n*m <=256.

Histogram

Histogram is a vector which gives the distribution of the grey-level values in the region of interest. If the image in the CogniSight memory plane is color and encoded with 3 bytes per pixel, the histogram reports the distribution of the average of Red, Green and Blue per pixel.

The number of bins in the case of a grey-level histogram is 256. The amplitude of the histogram is scaled by multiplying it by the ratio (255/N) with N, the total number of pixels in the region of interest.

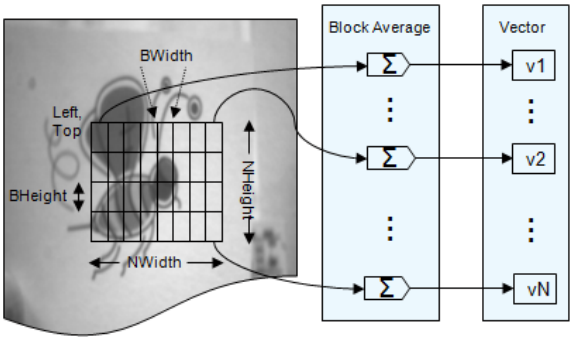
This feature vector gives an indication of the number of different shades in the region, the presence of noisy pixels and more.

Histogram Cumulative

The cumulative histogram is a mapping of the standard histogram which counts the cumulative number of pixels in all of the bins up to the specific bin.

SubsampleRGB

SubsampleRGB is a vector composed of the average Red intensities of blocks of pixels extracted from the region of interest, followed by the average Green and Blue intensities of these same blocks The blocks are surveyed in a raster displacement. The region must contain less than 81 blocks so the output vector fits on 81x3= 243 bytes. The blocks are all the same size, but not necessarily square. They are surveyed in a raster displacement and their average intensity is assembled into vector.



- The region with a size [NWidth, NHeight] is divided into up to 256 blocks of size [BWIDTH, BHEIGHT].
- The pixels of block #i are averaged to produce the i^{th} component of the signature vector.
- The relationship between the four parameters is :
    - NWIDTH= n*BWIDTH
    - NHEIGHT= m*BHEIGHT
    - n*m <=81

Histogram RGB

Histogram is a vector which gives the distribution of the Red, Green and Blue intensities in the region of interest. It is assembled as a series of 3 histograms of 85 bins each representing 85 bins for the Red, 85 bins for the Green and 85 bins for the Blue.

If the image in the CogniSight memory plane is monochrome and encoded with 1 byte per pixel, the histogramRGB will be a succession of 3 identical histograms since the Red, Green and Blue intensity of each pixel are the equal.

The number of bins in the case of a color histogram is 256. The amplitude of the histogram is scaled by multiplying it by the ratio (255/N) with N, the total number of pixels in the region of interest.

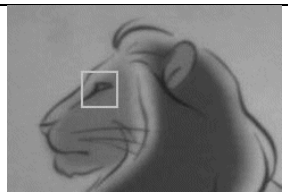
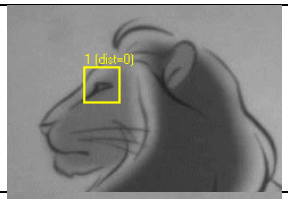
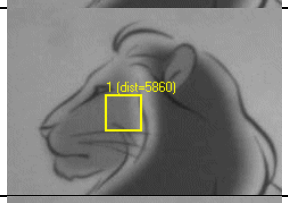
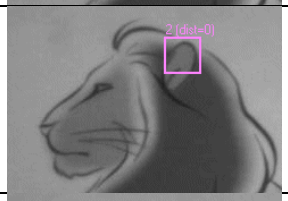
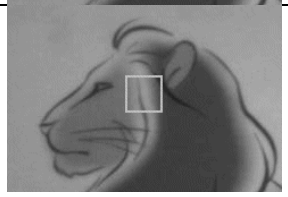
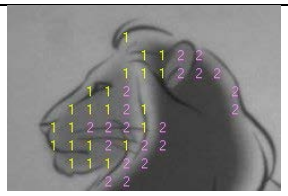
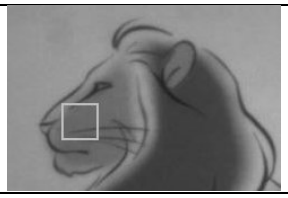
This feature vector gives an indication of the number of different shades in the region, the presence of noisy pixels and more.

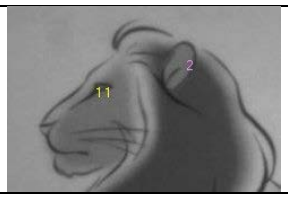
Histogram RGB Cumulative

The cumulative histogram is a mapping of the standard histogram which counts the cumulative number of pixels in all of the bins up to the specific bin.

APPENDIX B: TUTORIAL, HINTS AND TIPS

Supervised Object Learning

Description		Instructions	Comment
Size and position the ROI to include the eye of the lion and learn as category 1		SizeROI MoveROI LearnROI (1) Read Neurons=1	A first neuron is committed. It holds the model of the lion's eye and its associated category #1.
Recognize the ROI without moving it		RecognizeROI Read Distance =0 Read Category=1	The Distance 0 indicates that the signature of the ROI matches exactly the model of a neuron associated to category 1.
Move the ROI by a few pixels and recognize		MoveROI RecognizeROI Read Distance =5860 Read Category=1	The increase of the Distance value indicates that there is a drift between the signature of the ROI and the model of the closest (and only) neuron.
Move the ROI over the ear of the lion and learn as category 2.		MoveROI LearnROI (2) Read Neurons=2	A 2nd neuron is committed. It holds the model of the lion's ear and its associated category #2
Move the ROI in between the eye and the ear of the lion.		RecognizeROI Read Distance =0xFFFF	The Distance 0xFFFF indicates that the signature of the ROI is not recognized by either neuron which is the expected response.
Overview of the recognized objects		Set ROS SearchROS Read the VObjects and tag their position and category in the image	The 1s and 2s show respectively the locations recognized as the lion's eye and ear. The two neurons are over-generalizing and must be corrected.
Move the ROI over the nose of the lion and learn as background or category 0		MoveROI LearnROI(0) Read Neurons=2	No new neuron is committed but neurons 1 and 2 shrink their influence fields to stop recognizing the region as the eye or ear of the lion.

Overview of the recognized objects		Set ROS SearchROS Read the VObjects and tag their position and category in the image	The accuracy of the recognition is now satisfactory.

Using multiple feature extractions for object classification

The following example illustrates how to learn and recognize objects based on two different features for more robustness.

Let's take the example of character recognition.

Combining the use of a subsample vector and a histogram vector can help discriminate certain hand written digits.

In this case, two sub-networks of neurons will be trained to recognize the same input objects based on 2 feature vectors. The subsample can be assigned to the context 1 and the histogram to the context 2. The change of context must occur prior to the functions LearnROI, RecognizeROI and FindROSObject, MapROS. The change of feature is executed by the SetFeatParams function.

Example1: Learn a same example using 2 feature vectors

```
NeuroMem.GCR=1
SetFeatParams(FeatID1, norm1, minif1, maxif1, FeatParam11, FeatParam12, FeatParam13, FeatParam14)
LearnROI(X,Y,Cat1)
NeuroMem.GCR=2
SetFeatParams(FeatID2, norm2, minif2, maxif2, FeatParam21, FeatParam22, FeatParam23, FeatParam24)
LearnROI(X,Y,Cat1)
```

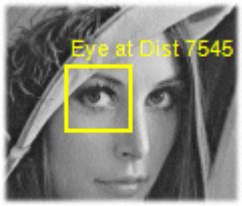
Example2: Recognize a same example using 2 feature vectors

```
NeuroMem.GCR=1
SetFeatParams(FeatID1, norm1, minif1, maxif1, FeatParam11, FeatParam12, FeatParam13, FeatParam14)
[Cat1out, Dist1out]=RecoROI(X,Y)
NeuroMem.GCR=2
SetFeatParams(FeatID2, norm2, minif2, maxif2, FeatParam21, FeatParam22, FeatParam23, FeatParam24)
[Cat2out, Dist2out]=RecoROI(X,Y)
If (Cat1out==Cat2out) printf("double score!")
```

What if an object appears at different scale factors?

The neurons assigned to a given context C and trained to recognize objects with a size W x H will be able to recognize the same object at different scales provided that the ratio of its primitive blocks remains the same.

If an object is taught using the following settings:
ROI size=rS
Block size=bS
The same neuron will recognize the object viewed at a scale N with the following settings:
ROI size= N x rS
Block size= N x bS

	This original image is used to teach an example of an eye.	ROI 32x32 Block 2x2
	Case 1: Image is zoomed out. The ROI and block size are both reduced using the same ratio of ½ ⇨ The feature vector is similar to the one extracted in the original image. ⇨ The neuron trained on the original image has a good chance to recognize this vector.	ROI 16x16 Block 1x1
	Case2: Image is zoomed out. The ROI and block size are kept the same ⇨ The feature vector encodes different information than the one extracted in the original image. ⇨ The neuron trained on the original image has less chance to recognize this vector.	ROI 32x32 Block 2x2

Dealing with objects of different sizes

The following example illustrates how to learn and recognize different objects or parts in a same image.

Let's take the example of a bottle inspection where
(1) The filling level of a bottle must be acceptable
- a. The region of the filling level is called ROI#1
- b. It is assigned to the Context value 1
- c. Examples of categories: very_low, low, medium, high, very_high

(2) The quality of the front label must be good
- a. The region of the filling level is called ROI#2
- b. It is assigned to the Context value2
- c. Examples of categories: slanted, scratched, folded, perfect, absent

The neurons can be used to learn and recognize objects with different sizes or a same size but different primitive blocks. In that case you simply need to assign a different context per object size or block ratio. The change of context must occur prior to the functions LearnROI, RecognizeROI and ScanROS.

Each time the type of the ROI is changed, the program must update the following registers: CM_WIDTH, CM_HEIGHT, CM_BWIDTH, CM_BHEIGHT,CS_ ROILEFT, CS_ROITOP, CM_GCR.

Example 1: Learn two examples of filling levels taken at different locations in a same image
SizeROI(Width1, Height1)
NeuroMem.GCR=1
MoveROI(Left1, Top1)
LearnROI(medium)
MoveROI(Left1', Top1')
LearnROI(Cat1)

Example 2: Learn two examples of labels taken at a same location in two different images
SizeROI(Width2, Height2)
MoveROI(Left2, Top2)
NeuroMem.GCR=2
GrabImage
LearnROI(perfect)
GrabImage
LearnROI(Cat2)

Example3: Inspect the two ROIs in a new image
SizeROI(Width1, Height1)
NeuroMem.GCR=1
MoveROI(Left1, Top1)
RecoROI(Distance1, Category1)
SizeROI(Width2, Height2)
MoveROI(Left2, Top2)
NeuroMem.GCR=2
RecoROI(Distance2, Category2)

Surface Inspection

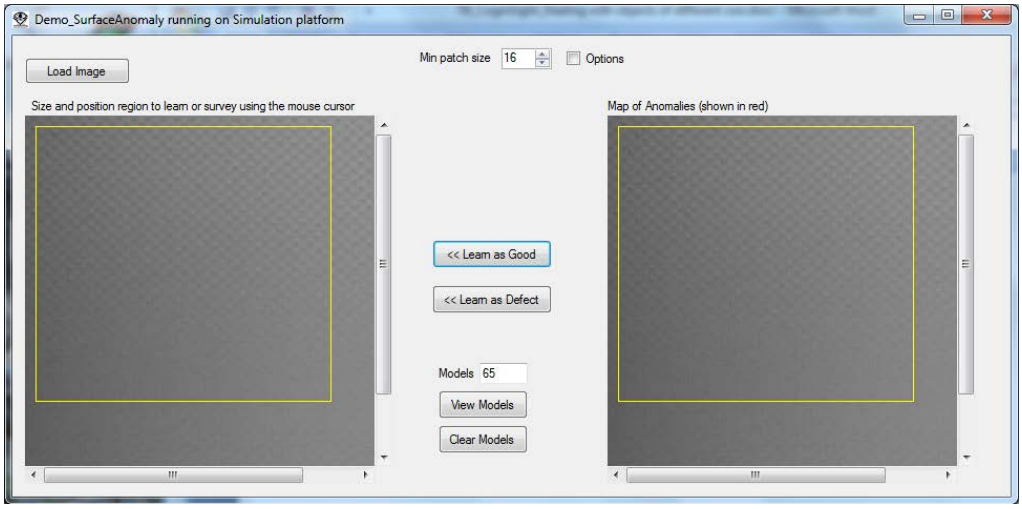
Texture learning is easy with the CogniSight engine. A region of interest can be divided into patches and the neurons will automatically learn the patches which are significant to describe the texture of the region.

In the example below, the surface inspected is solar glass which features a periodic bumpy pattern. Following is a series of patches of 16x16 pixels learned by the neurons. Assuming that a glass area with good quality is learned by taking examples of patches at all possible phase and assigning them the “Good” category, the content of the resulting committed neurons is a description of the good glass texture.

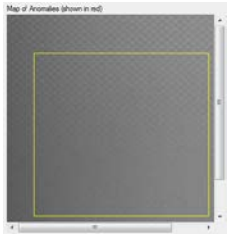
Example patches of 16x16 pixels

The user interface presented below is very simplistic but sufficient to illustrate how to develop a surface inspection system with the CogniSight technology. The area selected by the user and outlined in yellow has been learned as a “Good” texture and this has generated 65 models. The number of models depends on two settings of the learning operation: the value of the maximum influence field (MAXIF) of the neurons and the scanning step used to extract the sample patches from the region of interest.

- The higher the step and the smaller the number of samples.
- The smaller the MAXIF, the more models.

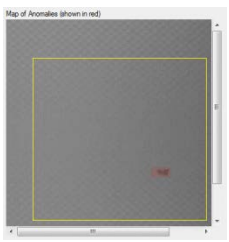


The image to the right is supposed to highlight the patches which are not recognized by the CogniSight engine because they do not match any of the 65 models. In this case, all learned patches are positively identified.



The same remark is true if the region of interest is moved around as shown in the image to the left.

This is made possible by learning the content of the region using a step of 1 or 2 which allows to generate representations of the patches of texture at many different phases:



If a new image is loaded and shows a significant defect, the neurons will not recognize the patches at the location of the defect. They appear highlighted in red in the Transform image.

In the event that a defect is not properly identified, a new region limited to patches covering the defect can be selected with the mouse cursor and learned as a Bad texture. This learning operation will have the effect to reduce the influence field of the neuron(s) recognizing the patches as good prior to learning them as counter examples.

More information…

General Vision has developed a complete Defect Detection System installed and tested over a glass float. It is revolutionary solution based on a scalable chain of V1KU cameras programmed with a same CogniSight engine for surface inspection and loaded with the same knowledge in their respective CM1K chips.